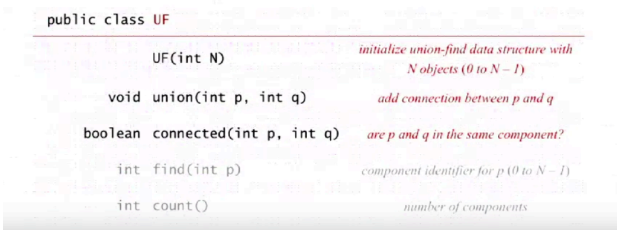
Union Find

- N = V= number of nodes;
  M = E= number of edges
- weighted (use sz) + path compression:
- individual operation:
  Union(p,q): lg*V
  ##improved by path compression, attach smaller tree to the larger—> tree height won't increase
  Root(p): lg*V
  ## when call root, rewind id[p] = id[id[p]] to trim tree height
  ifconnected(p,q):lgV
  ## this repeatedly calling root, so it trims the height on the way
- Whole operation:
  N node, for each edge need to call functions!
  initialization: V
  The whole algorithm: V + E lg* V (used E many union)
- 

DFS — both dir + undir

applications:
1. (dir+undir) Find all vertices connected to a given source vertex (union find) — reachability
2. (dir+undir)Find ONE path between two vertices
3. Topological sort
4. Directed cycle detection

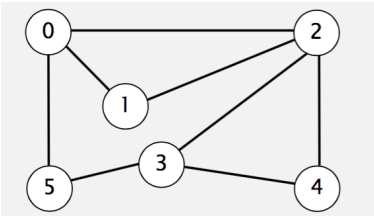
Connected or strongly connected components (dir+undir):
Time = O(sum of (vertex degree)) = O(2E)

General Time: O(E+V)

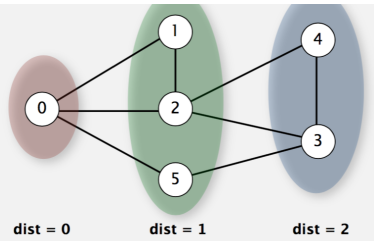
equivalent to Stack

BFS — both dir+undir

- shortest paths: time = O(E + V)
- when the actual problem could have a very long path, dfs may NOT work well. we use BFS



BFS



- BFS :shortest paths: time = O(E + V)
- eg LC 286: You are given a m x n grid initialized with these three possible values.
  1 - A wall or an obstacle; 0 - A gate; INF - empty room.
  Fill each empty room with the distance to its nearest gate. If it is impossible, fill it with INF.
- Number of vertices: V = m*n
  Each vertex connect at most 4 other vertices: E = 4*V
  So, BFS loops all vertices takes: O(E+V) = O(m*n)

BFS vs DFS both (undir + dig)

	DFS	BFS
Find a path between a, b	just a path	auto shortest
put unvisited	on a stack	on a queue (FIFO)
(undir)Time	O(E+V)	O(E+V)
Topological sort+	Yes (build a cycle detection within the algo)	NO
dir cycle detect	Yes	NO
find connected component	Yes for undir	Yes for undir
strong component	find reverse graph top order; then find graph cmpnts	NO